



Japan Technical Jamboree 76

# InnerSource Learning Path

株式会社東芝 ソフトウェア技術センター

小林 良岳 ( E-mail: [yoshitake.kobayashi@toshiba.co.jp](mailto:yoshitake.kobayashi@toshiba.co.jp) )

2021.3.11

CC BY-SA 4.0

# 本日の Takeaway

- コラボレーション開発 (**InnerSource**) の基礎と効果の理解
- それぞれの立場で何をすればよいか

# 発表の流れ

- 01 InnerSource とは？
- 02 参加してみよう
- 03 プロジェクトのまとめ役になろう
- 04 プロダクトオーナーとしての立ち回り
- 05 InnerSource の実践

# 01

## InnerSource とは？



## 1-1 こんな経験はありませんか？

- 一緒にやってるプロジェクトで他チームを待っていたら遅れた・・・
- バグを見つけて、通知したのに何時まで経っても直ってこない・・・
- 似たような機能を作ってしまい、両方メンテナンスしないといけない・・・

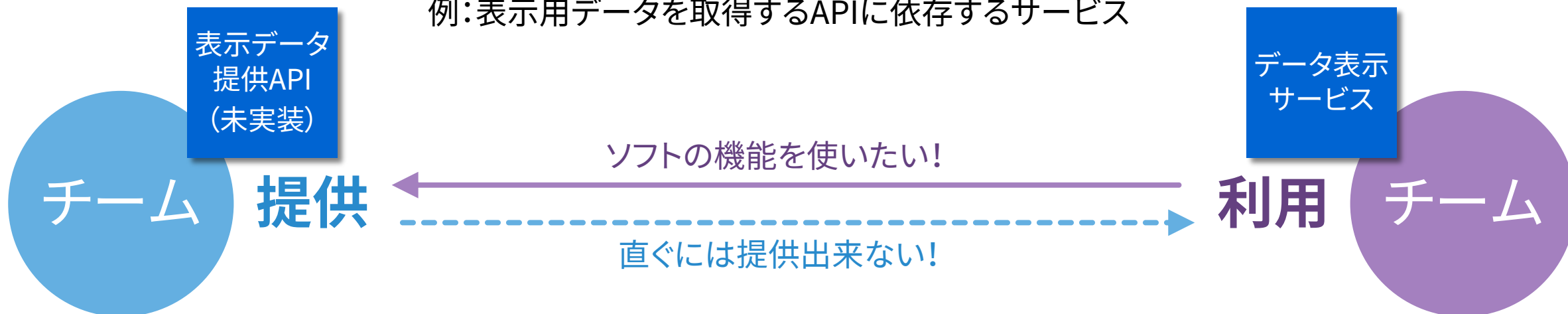


# 1-1 こんな経験はありませんか？

## ケース

同じ会社にある二つのチームが、別々のソフトウェア部品を提供する時、片方のチームのソフトウェアが、もう一方のチームのソフトウェアに依存する状況

例：表示用データを取得するAPIに依存するサービス



チームごとに事情は異なる

- ・機能の優先度
- ・開発スケジュール など

それでもリクエストが届かない  
ようなケースはどうするか？





## 1-1 こんな時、あなたならどうしますか？

### 1. 「静観」：黙っている

#### メリット

作業を最小限に  
することができる

#### デメリット

・要求された機能がいつまでたっても提供されない。

### 2. 「回避」：勝手にやる

#### メリット

要求機能が足りない部分を  
ローカルに変更・機能追加  
して補なえる

#### デメリット

・成果は同じ機能を必要としている他の利用者に提供されない  
・本来の役割範疇でないコードを長期的にメンテナンスしなければならない  
・会社全体として、同じ課題に対する重複したプロジェクトとコードを取得してしまう

### 3. 「圧力」：上層部を通してやらせる

#### メリット

必要な機能が手に入る  
(かもしれない)

#### デメリット

・圧力という開発に関係のない作業に注力しなければならない。  
・何度も使えるものでもなく発展しない。  
・チーム間や個人間の信頼を損なう。

## 1-2 InnerSource なら、まとめて解決できます！

### InnerSource を使うと**全ての欠点を解決し、効果が得られる**

- エンジニア同士が新しいさまざまな技術やバラエティに富んだ人々と一緒に仕事をする事ができる
- 共通の課題に対する解決方法を再利用することで、組織にとってより高い価値となることに集中して取り組むことができる

・・・でも、そもそも **InnerSource** って？



## 1-2 InnerSource とは？

- InnerSource は、**企業内のソフトウェア開発にオープンソースの実践とベストプラクティスと原則を適用**したものです。
- InnerSource ソフトウェアは、会社としてはプロプライエタリなものとなりますが、**内部にはオープンで、誰もが利用したり貢献したりできる**ようになります。

つまり・・・

**Open all artifacts!**  
**Welcome visitors!**

・・・を企業内で実践することです。

Dirk Riehle, Inner Source (Software Development), InnerSource Summit 2018

## 1-2 InnerSource で共創を実現

今まで



サイロごとに閉じている  
何をしているか自部門以外のことは知らない

目指す姿

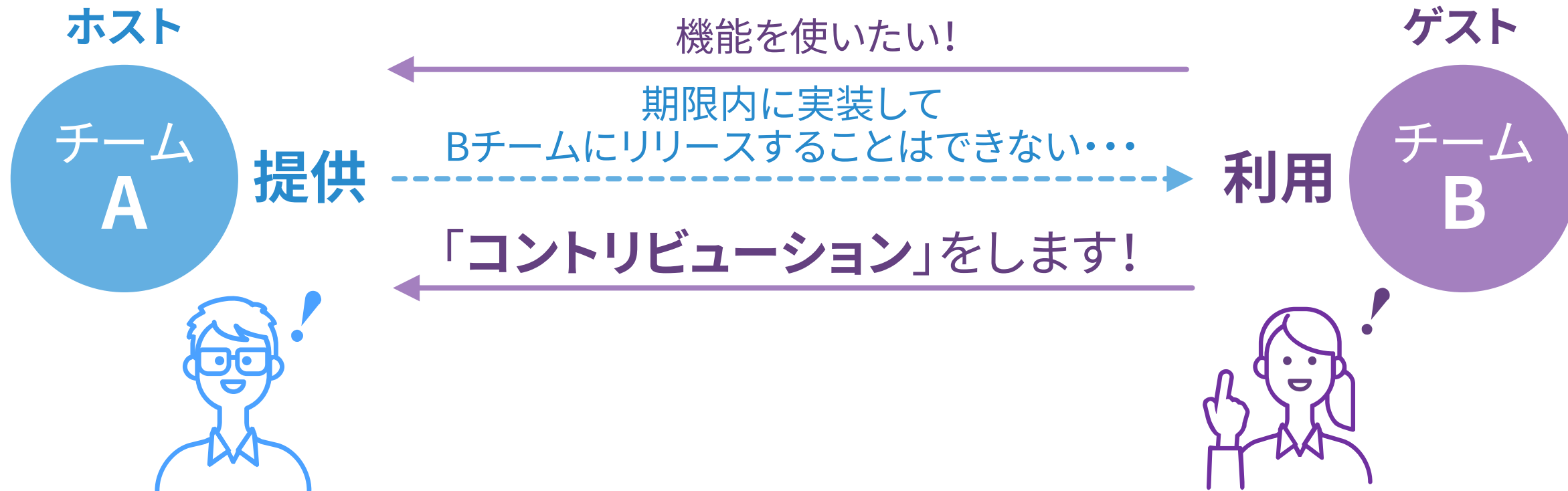


サイロを取り払い、コラボレーションが各所で発生！

## 1-3 どのように InnerSource は機能するのか？

### ケース

Aチームが提供するソフトウェアをBチームが利用する場合



## 1-3 InnerSource における役割

### チーム A〔ホスト〕

#### プロダクトオーナー

受け入れられるモノを仕分けする



ホストチームの家長的存在。  
受け入れるコントリビューションが  
どの機能かを決定

#### トラステッドコミッター (メンテナー)

さばき役。ゲストをサポートする



ホストチームを代表。  
コントリビューターが正しく貢献  
するために必要な指導やサポートを  
タイムリーに提供

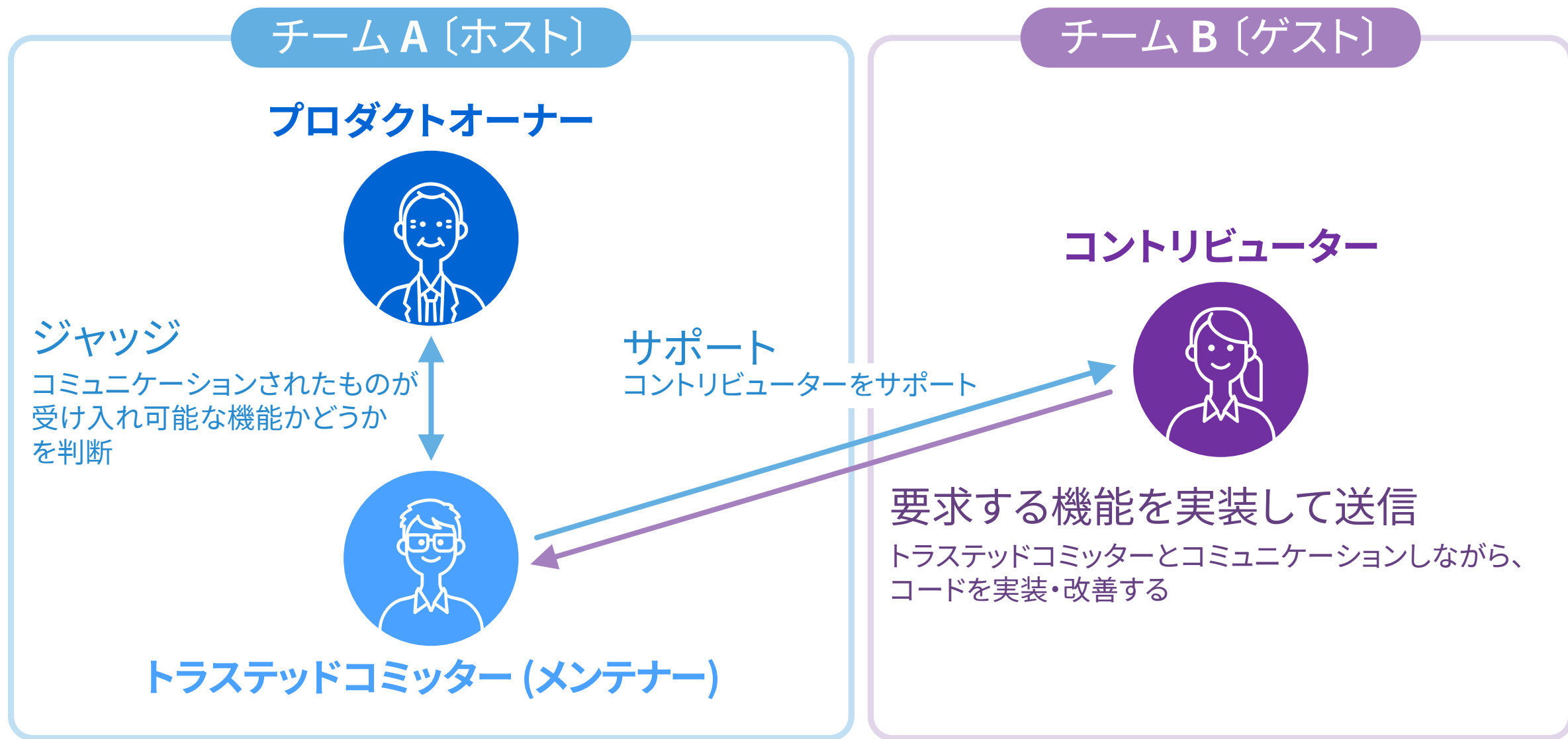
### チーム B〔ゲスト〕

#### コントリビューター



別チームの一員。  
ホストチームにコード等を投稿。

## 1-3 InnerSource における役割



## 1-4 InnerSource におけるメリット

### チーム A〔ホスト〕

より良い拡張性やサービスを顧客に提供することができる



### チーム B〔ゲスト〕

長期的にメンテナンスする責務を負うことなく、必要とする時間までに機能入手できる



誰もが利用可能な場所で共通課題に対する解決策を会社全体で共有できることは、会社にとっても有効



# 1-4 InnerSource の効果とは？

## 開発の利点

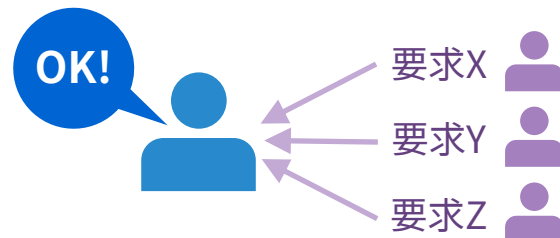
### チーム A〔ホスト〕

- ・ニーズが確定している機能を受け取れるので、良いプロダクトを作るための支援となる
- ・スケーラブルな戦略ができるようになる

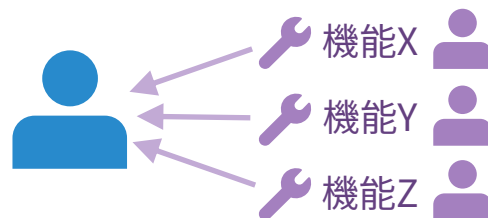
必要な機能が手に入る

品質向上

- ・必要とする時と場所にエンジニアリングの時間を有機的に投入することが可能



- ・全ての利用者との間で優先順位の調整を行うことができる



### チーム B〔ゲスト〕

長期メンテナンスの負担をせず、必要な時に機能要求を手に入れられる

必要な機能が手に入る

戦力倍増する

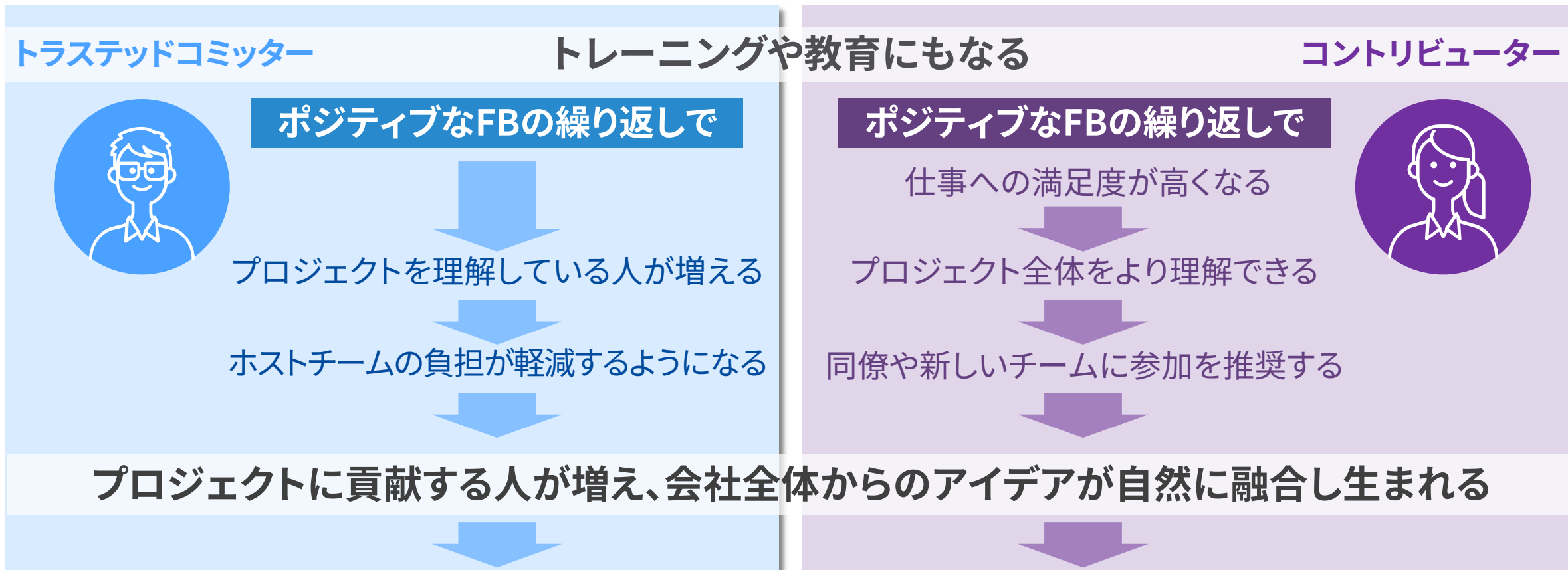
会社

リソース配分を最適化できる

余力ができ、他へ注力できる

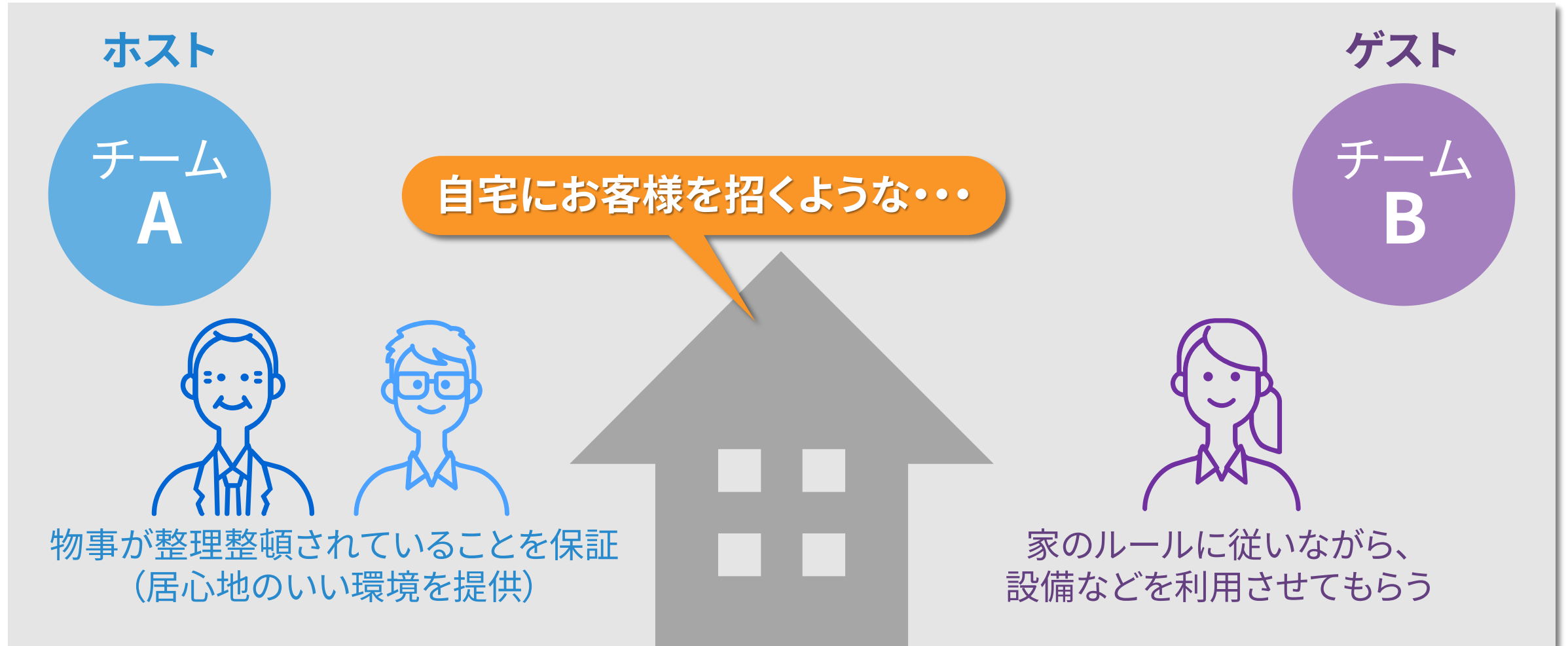
## 1-4 InnerSource の効果とは？

### 当事者の利点



従来の会社のサイロがなくなる

## 1-4 InnerSource での役割と心構え



## 1-5 InnerSource の原則

**オープン性** → 常にドアが解放されていて、受け入れ可能な状態

**透明性** → 意思決定プロセス、コード、ToDoなどが見えるようになってる状態

**優先的なメンターシップ** → (ゲストに対して) ありとあらゆるサポートを惜しまない  
(結果的に) 仲間になってくれる

**自発的なコードコントリビューション** → 共通の課題意識に基づき、お互いに今協力する

**一元管理された共有の場** で、  
誰もが目的のプロジェクトを見つけられるようにする

## 1-6 ここまでのまとめ

- **InnerSource** は、企業内のソフトウェア開発にオープンソースのベストプラクティスと原則を適用したものです。
- これは、提供側のチームが必要な機能要件を提供することができない時に、利用者に追加オプションを提供するものです。
- **InnroSource** の成功には、「**ホストチーム**」の**プロダクトオーナー**と**トラステッドコミッター**、そして「**ゲストチーム**」の**コントリビューター**が関わります。
- 効果的に行うと、**InnerSource** は両方の参加チームに多くの効果をもたらします。
- 効果的な **InnerSource** 実施の主要な原則は、**オープン性**、**透明性**、**自発的なコードコントリビューション**と**優先的なメンターシップ**です。

共通の場でオープンにコラボレーションしながら開発をすすめることで  
戦力を倍増していくこと

# 02

参加してみよう  
【コントリビューター】





## 2-1 InnerSource における役割

インナーソースには3つの役割がありましたね。

ここからは **コントリビューター** の話をします。

プロダクトオーナー  
受け入れられるモノを仕分けする



ホストチームの家長的存在  
受け入れるコントリビュー  
ターの機能かを決定

トラステッドコミッター (メンテナー)  
さばき役。ゲストをサポートする



ホストチームを代表。  
コントリビューターから貢献  
するために必要な指導やサポートを  
チーム内に提供



**コントリビューター**



別チームの一員。  
ホストチームにコード等を投稿。

もうひとつ重要なことは **共通の課題!**  
まずは、それについて話していきましょう。

## 2-1 共通の課題(一般的な課題)とは何か？

### 共通課題の候補になりそうなもの

#### 開発視点

- 共通サービス
- ○○プラットフォーム
- ○○フレームワーク
- ミドルウェア
- ライブラリ
- ツール
- 開発環境

#### コミュニティ 視点

- トップダウンで行われている組織横断活動・プロジェクト
- ボトムアップ活動
- (公認・非公認によらず) 社内○○コミュニティ

## 2-1 こんな経験はありませんか？〔再掲〕

- 一緒にやってるプロジェクトで他チームを待っていたら遅れた・・・
- バグを見つけて、通知したのに何時まで経っても直ってこない・・・
- **似たような機能を作ってしまい、両方メンテナンスしないといけない・・・**



## 2-1 共通の課題(一般的な課題)とは何か？

なにかを作ろうとするとき・・・



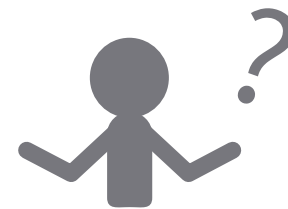
「なにか似たことを考えている人はいないかな？」と探してみる

つまり・・・

- 作る前に
- 考えていることを共有して
- 似たことを考えていたり、やっていたりする人がいないか見つけてみる

と言っても・・・

コントリビューションするモチベーションは何だろう？



## 2-1 皆さんも、このような事を考えるのでは…



それぞれのプロジェクトの進行ペースが違うよね!?

前にも伝えたのに、誰も動かなかったよ!



使おうとしたけど、バグばかりで使えなかったよ…(誰も直してくれないよ…)

そこで…

# InnerSource

**InnerSource** のプロジェクトではソースコードをチェックアウトして変更を加え、改良したバージョンを利用する自由があるんです!

## 2-1 コントリビューターになってみよう!

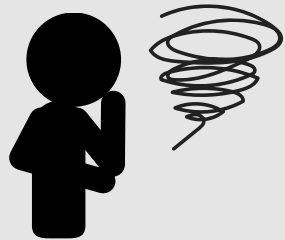
### メリット

プロジェクトのコピーを作成して、そこに変更を加えて、自分の手元に置く!

確かに、仕事は終わるけれど...

### デメリット

コピーしたコードをメンテナンスし、ホストチームが作成する新しいリリースのいずれにも、その変更を追従させないといけない



**InnerSource の自由を間違えると、  
こんな問題を抱えてしまう!**





## 2-1 コントリビューターになってみよう!

こんな問題を避けるためには...

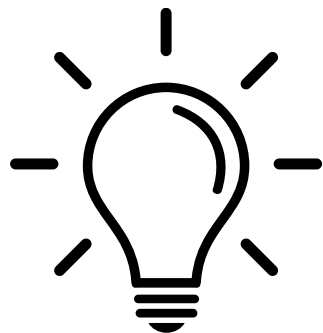


**自由に直せるのであれば、チェックアウト元に直接変更を加えよう!**



### マインドセットを変える

- 機能やバグの修正を待つ代わりに...
- 問題を回避する代わりに...



### 第3の解決策があることを認識する

- InnerSource プロジェクトでニーズを確認
- 共有されたプロジェクトに直接変更を加えます

## 2-1 コントリビューターになってみよう!

チェックアウト元に直接変更を加える行為をするのが  
**コントリビューター**です!



コントリビューターになれば、こんなメリットがあります!

- ・チェックアウト元のプロジェクトの専門家らから、より深い知識が得られる!
- ・本番でしか明らかにならないようなパッチの問題が早く明らかになって品質も上がる!

## 2-1 どんなことがコントリビューションできる？

- コンポーネントやコード、サービスを利用している際の問題を報告
- 使っていることの報告
- コードが期待通りに動作していないことを示すテストケース
- ドキュメントの改善
- 他のユーザのサポート
- バグのトリアージの支援
- ビルドの改善



**どんなことでも、ノウハウを共有するのは  
正しいコントリビューション！**

## 2-2 コントリビューターの心構え



### ソースコードを、どのようにコントリビューションすればよいか確認しよう

- ドキュメントがあれば、それを確認
- 情報が不足している時は、問い合わせできそうな人 (トラステッドコミッター) に確認

**それから、自分を売り込もう!**



## 2-2 コントリビューターになるときの“考え方ポイント”

- ① 「何か見つけたら」  
コントリビューションしてみよう!と考える
- ② コントリビューションの過程は、オープンにする
- ③ コメントをもらったら感謝する
- ④ もし、受け入れてもらえない時は・・・

## 2-2 コントリビュータになるときの“考え方ポイント”

### 1 「何か見つけたら」 コントリビューションしてみよう!と考える

コントリビューションする先のプロジェクトへの責任はなし!  
…という軽い気持ちで始めてみよう(※)

間違えた時には、ゴメンでOK



変更箇所へのホストから問い合わせには答えよう!  
(例:30日保証)



(※) 変更が正しいか確認して取り込むのはホスト側なので確認しているはず



## 2-2 コントリビュータになるときの ”考え方ポイント”

### 2 コントリビューションの過程は、オープンにしておこう！



他の人が同じところでハマる・・・  
経緯が分らないと、他の人も時間が掛かる・・・

オープンに議論することで、

- ・ **もっといい解決策の提案や**
- ・ **バグを直してくれたりなど**

があり、品質向上するかもしれない！



## 2-2 コントリビューターになるときの ”考え方ポイント”

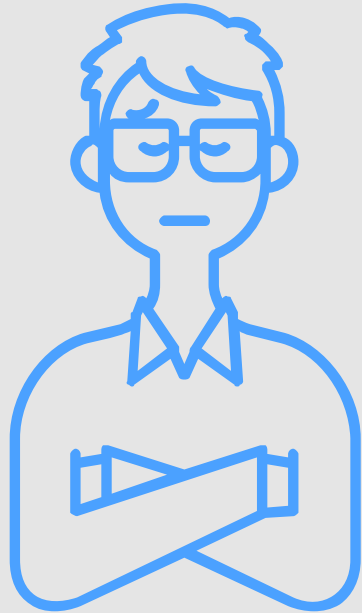
### 3 コメントをもらったら感謝する



## 2-2 コントリビュータになるときの ”考え方ポイント”

### 4 もし、受け入れてもらえない時は・・・

きっと何か問題があるはずなので(なぜ受け入れてくれないのか)  
別の解決策を(ホストチームと)一緒に考えよう!



## 2-3 コントリビューションを始めてみよう

基本は、次の3ステップ



コントリビューションの準備をする



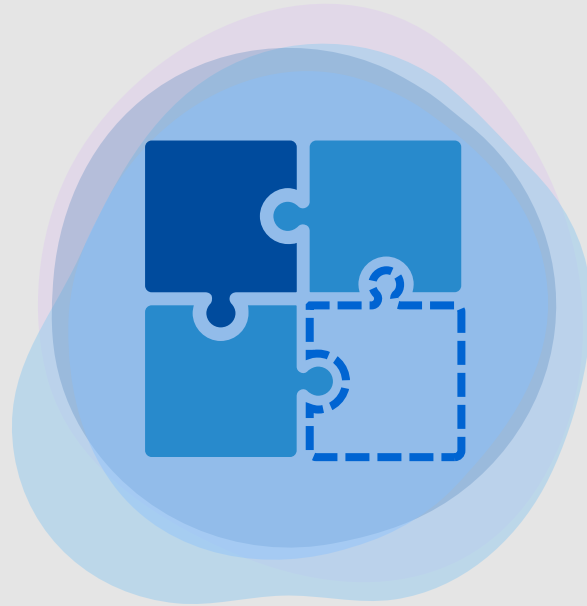
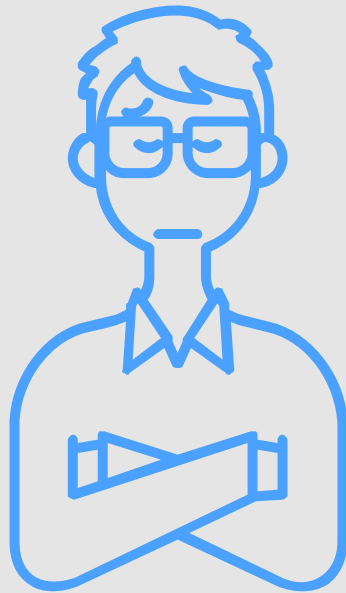
コードを作成する



ホストチームにプルリクエストを送る

## 2-3 コントリビューションの準備をする

考え方のポイントは  
ホストと「共通課題」になりそうかどうか



他に考えておくべきことは？

## 2-3 コントリビューションの準備をする

取り組む前に… 考える必要がある3つのポイント



**リードタイム**



**信頼関係の構築**



**変更を受け取ってもらうための戦略**



### リードタイム

- コミュニティに参加して馴染むまでの時間  
(開発環境、コード、雰囲気)
- 何回かコントリビューションすると、  
リードタイムは短くなります



### 信頼関係の構築方法

- 期待を裏切らないようにしよう！  
→ ちょっと時間がかかりそうなことは事前に伝えよう
- いろいろなコミュニケーション方法を使い分けよう！  
→ メールだけでは細かなニュアンスが伝わらない  
Web会議で話す、F2Fで合う機会を作ることも必要





### 変更を受け取ってもらうための戦略

- 何かコードを書く前に、方向性の確認をしておく  
(それが本当に必要なものなのか?)
- オープンな場で議論して、  
議論の過程が見えるようにする

## 2-3 コードを作成する

では機会が見つかったら…  
**コードを書こう!**

…その前に、壁になりそうなものは取り除こう!それを取り除くには?

- **相手のルールは守ろう**

コーディング規約

自分も検索できる  
ようになるよ

- **事を起こす前のコミュニケーションで、仲良くなろう**

コミュニケーションの過程は、オープンな場で行おう

次の人のためになるよ

- **自分で解決できそうにない部分は、できないと言おう**

## 2-3 プルリクエストを送ろう!



### 送るときのポイント

- できるだけ簡単に取り込んでもらえるように、送るときにはテストしよう!
- 説明をつけよう!



### マナー

一気に、大きな変更入れないようにしよう! 少しずつ順番に入れていこう

## 2-4 コントリビューションするメリットは？（個人）

### 単独で作業する代わりに、ホストチーム（アップストリーム）と作業するメリット

- ・ レビューと改善のサポートと指導を受けられる → 開発作業が大幅にスピードアップ

### 他の人と時間を過ごすメリット

- ・ 新しいツール、ソフトウェアについて実践的に学べる機会がある → 自分も後で使える
- ・ 複数の異なるプロジェクトに関係する → それぞれの良い点で活かせる
- ・ 自分のチームの境界を超えて人間関係と評判が広がる → いろんなところに意見を反映できる

### 自分が必要なところに取り組むことのメリット

- ・ 必要な部分へのコントリビューションでモチベーションを維持して取り組むことができる

## 2-4 コントリビューションするメリットは？（チーム）

### アップストリームで取り組むメリット

- メンテナンスのコストと時間をアップストリームプロジェクトに任せられる
  - アップストリームで新しいリリースごとテストが行われる
  - コントリビューションしたものの互換性のチェックはアップストリームで確認される

### 他チームのプロジェクトでアクティブなコントリビューターとして働くことのメリット

- プロジェクト方向性とスケジュールに発言権を持てる → **自チームに有益となる可能性**

### InnerSource の方法を使うメリット

- 「独自実装で自分で全てのバグ抱える」とこと、「既存実装で時間とお金を節約するが、自由に仕様が決められない」ということの間解を得られる
- 再実装と再利用とのバランスをとることが容易になる

## 2-4 コントリビューションするメリットは？（会社）

### 企業レベルで InnerSource を行うことのメリット

- チーム間のコラボレーション促進で、イノベーションを推進できる

### 会社全体で共有することのメリット

- 複数の実装が維持されなくなる
- 理解しやすく、再利用しやすくなり、結果としてモジュール化される
- より多くの視点でコードの変更が精査され、品質とセキュリティの向上につながる

### プロジェクトの透明性が高まることのメリット

- バスファクター（いなくなったら困る人）が1～2人という状況を回避できる
  - リスクのあるプロジェクトがわかりやすくなる
- ベストプラクティスやポジティブなイノベーションが、簡単に組織全体に広がる

**職場環境の改善が組織全体に広がりやすくなり、従業員が定着する**

# 03

プロジェクトのまとめ役になろう  
【トラステッドコミッター】

## 3-1 トラステッドコミッターの役割の紹介

プロジェクトの立ち上げ時には  
PJリーダー、開発チームリーダー 的な役割があるのでは？

InnerSource では



プロダクトオーナー



トラステッドコミッター



がそれらに該当します！



**InnerSource では、トラステッドコミッターの役割がとっても重要！**



## 3-1 トラステッドコミッターの役割の紹介

(トラステッドコミッター)  
**責任の範囲**

- ・品質の確保
- ・コミュニティの健全性維持
- ・コミュニティへの参入障壁を下げる
- ・コミュニティのレベル向上
- ・コミュニティのニーズを擁護



トラステッドコミッターは

**技術**

**+**

**コミュニティ（コラボレーションする人全員）**

の両方の面倒をみる

## 3-2 品質の確保について

### ポイント

TCは技術や品質に関する意思決定または意思決定の調停を行う権利を持つ

### やり方

- 品質基準を決める
- コードのピアレビューを行って受け入れ可否を判断する
- (修正する代わりに) レビューの結果をフィードバックする
- コードのマージを行う
- 時間がかかる場合、新しい要求が生じる場合などは、スケジュールの修正を行う

何故修正を行わないの？

コミュニティ(とコード)の寿命を伸ばすための将来への投資

…だからです

### ● 健全なコミュニティとは？

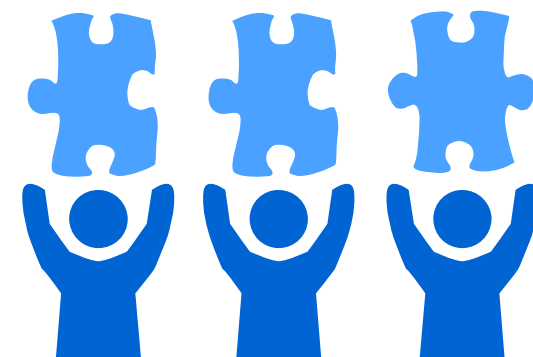
- コントリビューターがソフトウェア開発に時間を費やすことができる
- コントリビューターが能力を高めることができるようになっている

→ 結果として継続的に成長



### ● なぜコミュニティができるのか？

- 共通の課題や目的があるから
- コミュニティーの他のメンバーと一緒に仕事をするのが楽しいから



### 3-3 コミュニティの健全性維持

#### トラステッドコミッターが行うことコミュニティ健全性維持のための行うこと

- コミュニティ共通の課題を明確に表現して宣伝する  
→ **つまりはマーケティング**
- コントリビューションを贈り物として扱い、称賛する
- メンタリングを行う  
(何故改善する必要があるか、どう改善する必要があるか、理由と方向性を示す)
- 必要に応じて、行動規範を作成して実行する
- 双方が定期的にお互いを知り合う機会を提供する
- 紛争を平和的に解決する機会を作る



**コントリビューターが歓迎されて感謝される環境を作るように努力する**

## 3-4 コミュニティメンバーのレベルアップ

### コミュニティメンバーのレベルアップは何故必要？

- コミュニティの維持 (利用者やコントリビュータから将来トラステッドコミッターを発掘)
- コミュニティの活性化 (スピードアップ、アウトプット品質の向上)

### トラステッドコミッターがコミュニティメンバーのレベルアップのために行うこと

- 製品やコミュニティのマーケティング
- 貢献する機会の創出 (例: ドキュメンテーション、テスト自動化)
- **メンタリング(コードを受け入れ可能なレベルにする指導)**
- 成長する可能性のあるコントリビューターを発掘
- コントリビュータ、TC双方の学習と成長 (例: メンタリング能力向上)
- トップタレントの維持



**貢献する機会を伝え、コントリビューターを支援したり指導する**

より優れたソフトウェアをより早く作成するコミュニティ能力向上につながる

## 3-5 コミュニティへの参入障壁を下げる

何故、参入障壁を下げないといけないか？

- (会社の中なので) 潜在的なコントリビューターの数が少ない中で、見つけやすくする
- 他の仕事もある中で、なるべく手間を掛けずにコントリビューションできるようにする



## 3-5 コミュニティへの参入障壁を下げる

トラステッドコミッターが参入障壁を下げるために行うこと

参加までに迷わない道筋を用意する！

わかる

使える

繋がれる

### README を用意

- リポジトリに何があるのか、何に使えるのかを説明
- ライセンスに関する情報 (InnerSourceライセンス)
- ソフトウェアの入手方法、ビルド方法、テスト方法、使用方法についての詳細な指示を提供

### CONTRIBUTING を作成して コントリビューターに 何が期待されているか を説明

- バグレポートや機能リクエストはどうやって提出すればいいですか？
- 質問がある場合は誰にどのように連絡すればいいのか？
- コードスタイル、分岐、コミットメッセージの規約は？
- 貢献の「完了」の定義は？
- 貢献を管理するプロセスステップとは？
- 貢献が承認された後、貢献したコードをサポートするという点ではどのようなことが期待されていますか？
- 行動規範とは何ですか？

### オプション

- HELPWANTED を作成して、どのようなものが足りていないか明示する  
(例：アートワーク、イベントの企画、要求機能)

## 3-5 コミュニティへの参入障壁を下げる

トラステッドコミッターが参入障壁を下げるために行うこと

参加までに迷わない道筋を用意する！

わかる

使える

繋がれる



できるだけ多くの人が貢献できるように

- ドキュメントを用意して基本的な質問に答えられるようにする
- 貢献することがポジティブになる雰囲気を作り出す



## 3-6 コミュニティのニーズを擁護する

### トラステッドコミッターが行うこと

- 組織との信頼関係を構築する
- コミュニティの利益と社内のソフトウェアの長期的な健全性のために行動する
- 技術的なリスクだけでなく、コミュニティに関連するリスクをマネージャーに伝える
- コミュニティや個々のコントリビューターの仕事を外から評価(称賛)する
- (必要に応じて)コントリビューターのマネージャーと話し合いを行うなどのロビー活動をする



**個々の投稿者の利益とコミュニティ全体の利益を擁護して  
コミュニティの健全性を維持することで、  
コミュニティと組織の両者の信頼関係を構築**

## 3-7 トラステッドコミッターになる

### トラステッドコミッターになるにはどうすればいいの？

- 開発リーダー(クラス)になる
- コミュニティ活動で、周りから認められる

### トラステッドコミッターになるために行うことは？

- プロジェクトについて深い技術的能力を持つ
- プロダクトオーナーやマネージャーと効果的にコミュニケーションをとれるようになる
- コントリビューターをレベルアップさせる意欲と忍耐力を示す
- 感情的ならずに意見を受け止められるようになる
- コーディングより、コミュニケーション・メンタリングに注力する

### トラステッドコミッターになるための心構えは？

- 自発的に引き受ける



会社や  
マネジメント層で  
考えること

トラステッドコミッターを1人持つか複数持つか (規模やリスクで判断する)  
例: トラステッドコミッターの仕事を分担するローテーション・システム

# 04

## プロダクトオーナー としての立ち回り 【プロダクトオーナー】



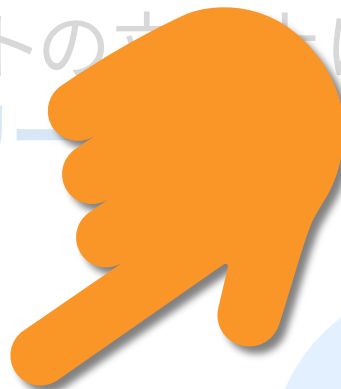
## 4-1 プロダクトオーナーとは？

プロジェクトの立ち上げ時には  
PJリーダー、開発チームリーダーなどの役割がありますよね？

InnerSource では



プロダクトオーナー



トラステッドコミッター

がそれらに該当します！



InnerSource における**プロダクトオーナー**とは？

## 4-1 プロダクトオーナーとは？

### 「中間管理職（ミドルマネージャー）」と呼ばれる人？

- それぞれの組織において、上層部のビジョンの執行に責任を負う人
- 予算管理や執行をする人
- 部・課・プロジェクト・チームなどの単位の責任者

### InnerSource におけるプロダクトオーナー

- 方向性に対して責任を持つ人（※注：開発ではなく）

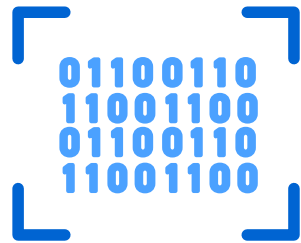


トラステッドコミッターとプロダクトオーナーが  
同一人物の場合もある



## 4-2 InnerSource のプロダクトオーナーとして最初に心がけること

オープンコード



ソースコードの公開

オープンプランニング  
・オープンドキュメント



計画とそれに関する文書

## 4-2 InnerSource のプロダクトオーナーとして最初に心がけること

### オープンコード

- コードを会社の全員が見えるようにする
- 他の開発者が他のコードベースでコントリビューションでき、それを受け入れるプロセスがある

### 効果

- 修正や機能実装を待ったり、エスカレーションしたり・されたりする必要がなくなる
- それぞれの優先順位に基づいて、コントリビューションできるようになる

## 4-2 InnerSource のプロダクトオーナーとして最初に心がけること

### オープンプランニング・オープンドキュメント

- ・標準化された方法(同じ場所)でプランニングプロセスやドキュメントを公開すること

#### 効果

- ・プロジェクトや製品ごとにドキュメントがどこにあるのかわかる(検索できる)
- ・見つけてもらえ、利用してもらえ、フィードバックがもらえる
- ・他のチームが何に取り組んでいるのか、現在何を優先しているのかを知ることで、より効果的にチーム間の調整を行うことができる
- ・議論の履歴を元に、優先順位が変更される理由などが明確になり、チーム同士の信頼関係構築につながる
- ・コラボレーション可能なチームを簡単に見つけられる
- ・お互いの開発文化の違いを知ることができる



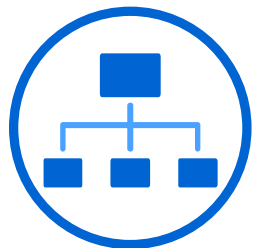
## 4-3 プロダクトオーナーの役割と責任



トラステッドコミッターの選抜とサポート



他のプロダクトオーナーとの交渉



環境の整備（仕事をすすめる上での）



社内マーケティング



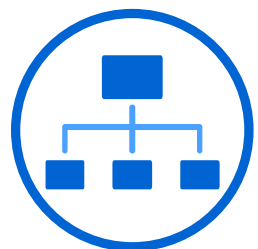
### トラステッドコミッターの選抜とサポート

- 見積などの協力を依頼
- トラステッドコミッターのメンタリング



### 他のプロダクトオーナーとの交渉

- 制約（人・時間・資金など）の中で、  
自チームの意見を代弁して理解してもらう
- 個々のチームのプロセスを尊重しつつ、  
必要に応じて他チームにも指導を行う



### 環境の整備(仕事をすすめる上での)

- チームに対するドキュメンテーションの時間の確保
  - ・ オープンドキュメンテーションに時間を割くことへの理解
  - ・ UXやUIの標準、API標準、テスト要件等の明確化依頼
- 開発者が安心して活動できる環境の整備
  - ・ 標準的な開発ツールの整備の許可



### 社内マーケティング

- コードを利用してもらえそうなプロジェクトの発掘  
(似たような機能開発をしているプロジェクトの発掘)
- コードを提供してもらえそうなプロジェクトの発掘
- ベストプラクティスや失敗談の共有
- Codeathon / Hackathon開催など、  
さまざまな種類のアナウンス

## 4-4 InnerSource のプロダクトオーナーになる利点

- コラボレーションで、  
**より少ないリソースでより多くのことを達成**できる実感が得られる
- チームの**冗長性が向上**する
- プロセスを公開することで**政治的な問題にも対処**できる
- トラステッドコミッターのサポートなど、  
**新たな役割と責任を得る**ことができる
- 社内マーケティング**スキルを身につけるチャンス**が得られる
- 仕事に対する**評価が高くなる**

# 05

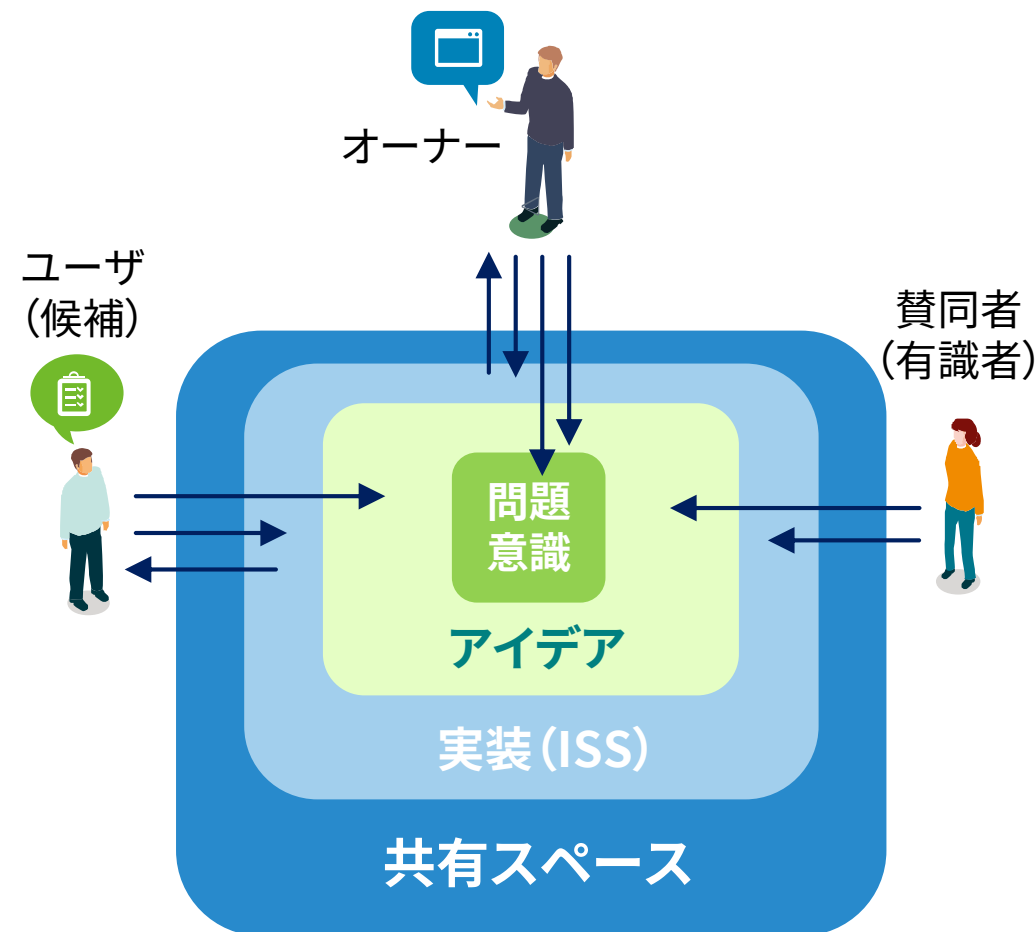
## InnerSource の実践



## 5-1 InnerSource 実践のための準備

### InnerSource に必要なもの（「共通」の問題・課題）

1. 問題意識を共有する・発見する
2. 共通の問題意識をもつ人達で意見交換する
3. 課題を解決する
4. (事業領域に関連する部分を作る)
5. 4の途中で出てくる共通部分の問題は共有する
6. ノウハウが共有され、新しいアイデアが出る  
(可能性がある)
7. 1に戻る





## 5-2 「共通課題」を見つけるには？

### ●まずは身近なところから

- ・ 同じ部門・組織の中から
- ・ 同じ事業ドメインの中から
- ・ 会社全体で

### ●どのようなタイミングで考えるのが良いか？

- ・ 新しいプロジェクトを始めるとき
- ・ 要件定義やシステム設計するとき
- ・ リファクタリングするとき

## 5-3 「共通課題」の解決にあたり重要なこと

オープン性

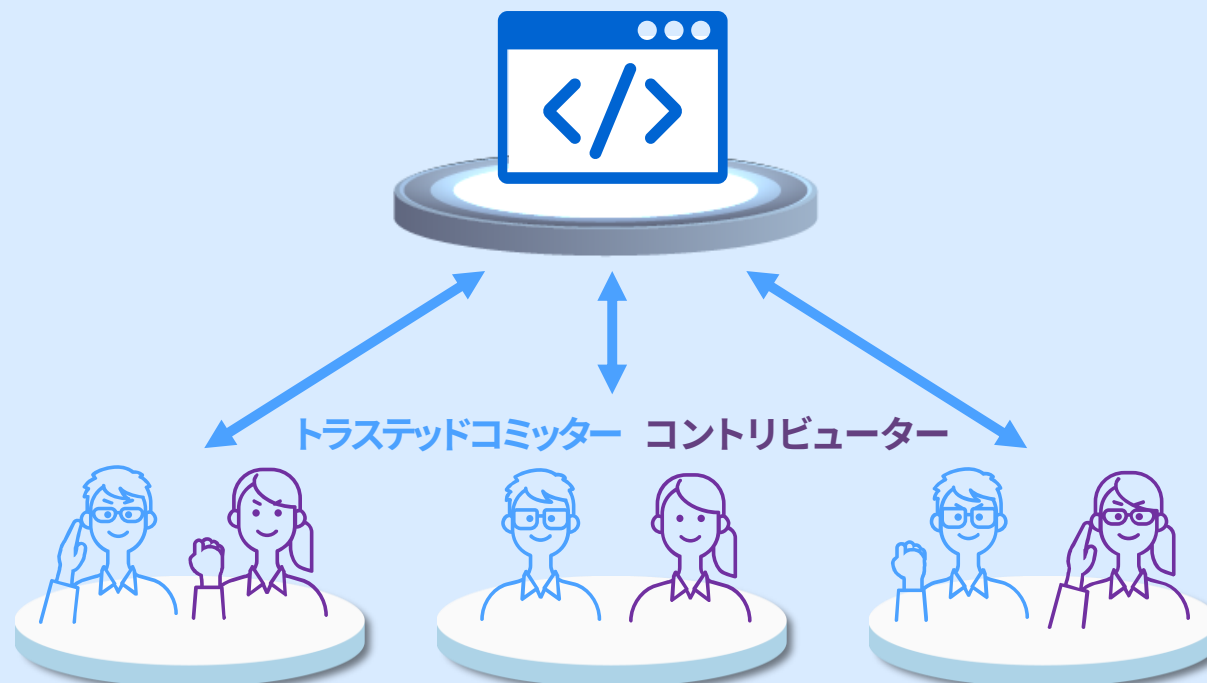
透明性

優先的な  
メンターシップ

自発的な  
コントリビューション



共有された場所を実施



## ● InnerSource Learning Path

- <http://innersourcecommons.org/resources/learningpath/>
- <https://github.com/InnerSourceCommons/InnerSourceLearningPath>
  - Copyright: Johannes Tigges (English), Yoshitake Kobayashi and Jun Ohtani (Japanese translation)
  - License: CC-BY-SA 4.0

## ● InnerSource Commons

- <http://innersourcecommons.org/>

**TOSHIBA**